Amendments to the Specification:

[0005] A method and system for executing 32-bit flat address programs during a System Management Interrupt is provided. In one embodiment, the SMM technique provides a 16-bit SMI routine that is given control when an SMI occurs. That routine initially saves the state of the processor and then executes an instruction to switch to protected mode. When in protected mode, the routine transfers control to 32-bit code. The 32-bit code uses a global descriptor table ("GDT") that is different from that used by the interrupted operating system. When the 32-bit code completes, it restores the saved processor state and returns from the interrupt by executing an RSM ("Resume from System Management Mode") instruction.

[0006] In one embodiment, the SMI routine invokes a 32-bit operating system kernel that is developed using standard 32-bit Windows development tools. This kernel may be loaded into SMM memory by the BIOS during system initialization. SMIs are caused to occur periodically (e.g., every 16 milliseconds) by programming the power management timers on external control circuitry implemented in the "chipset" components on the motherboard. An SMI is generated when an SMI input line to the processor is set by the chipset. The 16-bit SMI routine loaded from the BIOS receives control when these SMIs occur, saves the processor state, switches into 32-bit flat protected mode, and transfers control to a 32-bit operating system kernel. The kernel runs entirely in 32-bit protected mode, and is capable of loading and running standard Windows NT Portable Executable programs within the SMI context. (Of course, these programs would typically only use APIs ("Application Program Interface") provided by the proprietary 32-bit kernel.) The 32-bit code executed during an SMI can run programs transparently to the foreground operating system and can run them even while the foreground operating system has crashed or stopped responding (e.g., Linux Panic, or Windows NT Blue Screen). This SMM technique enables various applications to be developed to execute during an SMI.

The applications may include a remote console, remote boot, remote diagnostics, remote restart, and debugging.

[0007] Figure 1 is a flow diagram illustrating the processing of the SMI routine's transition all the way to the 32-bit flat address mode environment. In block 101, the routine saves the state of the processor, which may include the saving of the floating point register. In block 102, the routine switches to protected mode by initializing the Global Descriptor Table Register ("GDTR") to point to a new GDT, setting a bit in the processor's CR0 ("Control Register 0") register to switch to protected mode, and finally loading segment registers with protected mode selectors mapped as 32-bit segments by the new GDT. In block 103, the routine transfers control to the 32-bit application code by a NEAR CALL instruction to the pre-loaded 32-bit kernel dispatcher entrypoint. In block 104, the routine restores the saved state of the processor. In block 105, the routine returns from the interrupt by executing the RSM instruction.

[0008] In one embodiment, the BIOS ("Basic Input Output System") would preload the 32-bit SMM kernel into SMRAM and/or other physical memory reserved for SMM activities early on during Power On Self Test (POST). The kernel would be retrieved from a compressed bit stream stored in the same Flash ROM as the BIOS, to minimize its impact on the overall size of the firmware footprint in the system. Any 32-bit flat address programs that would need to run in the environment would also be stored as compressed bit streams stored in the same Flash ROM or another Flash ROM in the system, and be subsequently loaded by the 32-bit kernel once it had received control in System Management Mode via an SMI.